UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/730,900	12/10/2003	Jonathan Maron	19111.0117	5194

68009 7590 11/28/2007

BINGHAM MCCUTCHEN, LLP
2020 K STREET, NW
BOX IP
WASHINGTON, DC 20006

EXAMINER	
CHEN, QING	

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
11/28/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/730,900	MARON, JONATHAN	
	Examiner	Art Unit	
	Qing Chen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *05 September 2007*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,3-10,12-16,18-25,27-31,33-40 and 42-51* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,3-10,12-16,18-25,27-31,33-40 and 42-51* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *25 July 2007* is/are: a)☐ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is in response to the amendment filed on September 5, 2007.

2. **Claims 1, 3-10, 12-16, 18-25, 27-31, 33-40, and 42-51** are pending.

3. **Claims 1, 5, 6, 14-16, 18, 20-22, 27, 29-31, 33, 35-37, 42, 44, and 45** have been amended.

4. **Claims 2, 11, 17, 26, 32, and 41** have been cancelled.

5. **Claims 46-51** have been added.

6. The objections to the drawings are withdrawn in view of Applicant's amendments to the drawings.

7. The objection to the abstract is withdrawn in view of Applicant's amendments to the abstract.

8. The objections to the specification are withdrawn in view of Applicant's amendments to the specification.

9. The objections to Claims 5, 14, 20, 29, 31, 35, and 44 are withdrawn in view of Applicant's amendments to the claims.

10. The 35 U.S.C. § 112, second paragraph, rejections of Claims 1, 3-15, 18-30, 33-37, and 42-45 are withdrawn in view of Applicant's arguments and amendments to the claims. The 35 U.S.C. § 112, second paragraph, rejections of Claims 2, 11, and 26 are withdrawn in view of Applicant's cancellation of the claims. However, the 35 U.S.C. § 112, second paragraph, rejections of Claims 5, 6, 14, 15, 20-25, 29, 30, 35, 36, 44, and 45 are maintained in view of Applicant's arguments and further explained below.

11. The 35 U.S.C. § 101 rejections of Claims 31-45 are withdrawn in view of Applicant's amendments to the claims.

Response to Amendment

Drawings

12. The drawings are objected to as failing to comply with 37 CFR 1.84(p)(5) because they include the following reference character(s) not mentioned in the description:

- Reference numbers 103 and 105 in Figure 1.

Corrected drawing sheets in compliance with 37 CFR 1.121(d), or amendment to the specification to add the reference character(s) in the description in compliance with 37 CFR 1.121(b) are required in reply to the Office action to avoid abandonment of the application.

Any amended replacement drawing sheet should include all of the figures appearing on the immediate prior version of the sheet, even if only one figure is being amended. The figure or figure number of an amended drawing should not be labeled as "amended." If a drawing figure is to be canceled, the appropriate figure must be removed from the replacement sheet, and where necessary, the remaining figures must be renumbered and appropriate changes made to the brief description of the several views of the drawings for consistency. Additional replacement sheets may be necessary to show the renumbering of the remaining figures. Each drawing sheet submitted after the filing date of an application must be labeled in the top margin as either "Replacement Sheet" or "New Sheet" pursuant to 37 CFR 1.121(d). If the changes are not

accepted by the Examiner, the Applicant will be notified and informed of any required corrective action in the next Office action. The objection to the drawings will not be held in abeyance.

Specification

13. The title of the invention is not descriptive. A new title is required that is clearly indicative of the invention to which the claims are directed.

Claim Objections

14. **Claims 1, 3-10, 12-15, 18, 27, 33, 42, 46, and 50** are objected to because of the following informalities:

- **Claim 1** recites the limitation "creating an event handler for method nodes." Applicant is advised to change this limitation to read "creating an event handler for a method node" for the purpose of keeping the claim language consistent throughout the claims.
- **Claims 3-10, 12-15, 46, and 50** depend on Claim 1 and, therefore, suffer the same deficiency as Claim 1.
- **Claim 12** contains a typographical error: Claim 12 should depend on Claim 1, not Claim 11.
- **Claim 18** contains a typographical error: Claim 18 should depend on Claim 16, not Claim 17.
- **Claim 27** contains a typographical error: Claim 27 should depend on Claim 16, not Claim 26.

- **Claim 33** contains a typographical error: Claim 33 should depend on Claim 31, not Claim 32.
- **Claim 42** contains a typographical error: Claim 42 should depend on Claim 31, not Claim 41.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

15. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

16. **Claims 1, 3-10, 12-16, 18-25, 27-31, 33-40, and 42-51** are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claims 1, 16, and 31 contain the trademark or trade name ENTERPRISE JAVA BEAN. When a trademark or trade name is used in a claim as a limitation to identify or describe a particular material or product, the claim does not comply with the requirements of the 35 U.S.C. 112, second paragraph. *Ex parte Simpson*, 218 USPQ 1020 (Bd. App. 1982). The claim scope is uncertain since the trademark or trade name cannot be used properly to identify any particular material or product. A trademark or trade name is used to identify a source of goods, and not the goods themselves. Thus, the use of a trademark or trade name in a claim to identify or describe a

material or product would not only render a claim indefinite, but would also constitute an improper use of the trademark or trade name.

Claims 3-10, 12-15, 46, and 50 depend on Claim 1 and, therefore, suffer the same deficiency as Claim 1.

Claims 18-25, 27-30, 47, and 51 depend on Claim 16 and, therefore, suffer the same deficiency as Claim 16.

Claims 33-40, 42-45, 48, and 49 depend on Claim 31 and, therefore, suffer the same deficiency as Claim 31.

Claims 5, 6, 14, 15, 20-22, 24, 29, 30, 35, 36, and 44-51 recite the limitation "the markup language description." There is proper explicit antecedent basis for the limitation. However, the claims are rendered indefinite because it is unclear to the Examiner whether "the markup language description" is referring to "an Extensible Markup Language description" or not. For example, Claim 5 depends on Claim 4, which recites "an Extensible Markup Language description." Claim 4 ultimately depends on Claim 1, which recites "a markup language description." Based on this dependency order of the claims, "the markup language description" in Claim 5 has antecedent basis from either "an Extensible Markup Language description" or "a markup language description." In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "a markup language description" for the purpose of further examination.

Claim 23 depends on Claim 22 and, therefore, suffers the same deficiency as Claim 22.

Claim 25 depends on Claim 24 and, therefore, suffers the same deficiency as Claim 24.

Claim Rejections - 35 USC § 103

17. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

18. **Claims 1, 3-10, 12-16, 18-25, 27-31, 33-40, and 42-51** are rejected under 35 U.S.C. 103(a) as being unpatentable over **US 6,925,631 (hereinafter Golden)** in view of **US 6,754,659 (hereinafter Sarkar)**, and further in view of **US 2003/0158832 (hereinafter Sijacic)**.

As per **Claim 1**, Golden discloses:

- creating an event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.")*;

- registering the event handler *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.")*;

- parsing the markup language description and invoking the registered event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.")*; and

- generating output code using the invoked event handler *(see Column 14: 45-46, "... the taglet document is written to the output stream 15.")*.

However, Golden does not disclose:

- receiving an archive file to be deployed, wherein the archive file is an Enterprise Java Bean™ archive file;

- introspecting an input class included in the Enterprise Java Bean™ archive file to generate information relating to the input class; and

- generating a markup language description of the input class based on the generated information relating to the input class.

Sarkar discloses:

- receiving an archive file to be deployed, wherein the archive file is an Enterprise Java Bean™ archive file *(see Column 6: 50-55, "Then, at step 606, the Access Bean object AB701 retrieves environment information about generic EJB 720, locates it, and creates an instance of the generic EJB 720 using standard EJB API calls.")*; and

- introspecting an input class included in the Enterprise Java Bean™ archive file to generate information relating to the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include receiving an archive file to be deployed, wherein the archive file is an Enterprise Java Bean™ archive file; and introspecting an input class included in the Enterprise Java Bean™ archive file to generate information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

Sijacic discloses:

- generating a markup language description of the input class based on the generated information relating to the input class *(see Paragraph [0119], "Once a Java class that implements the ISimpleWorkPerformer interface is created and compiled, an XML description file for the class is defined. The XML description file specifies the environment, input, and output parameters that the class uses. In addition, the XML file specifies some optional design parameters that may control the custom activity's appearance in the process builder 391.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sijacic into the teaching of Golden to include generating a markup language description of the input class based on the generated information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to describe a class of data objects using tags *(see Golden – Column 3: 40-48)*.

As per **Claim 3**, the rejection of **Claim 1** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain*

the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

As per **Claim 4**, the rejection of **Claim 3** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 5**, the rejection of **Claim 4** is incorporated; and Golden further discloses:

- creating a Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in a markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The*

event handler interface DocumentHandler is called whenever an element is found in the input stream.").

As per **Claim 7**, the rejection of **Claim 1** is incorporated; and Golden further discloses:

- creating a plurality of event handlers for a method node found in the markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 8**, the rejection of **Claim 7** is incorporated; and Golden further discloses:

- registering each of the plurality of event handlers *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 9**, the rejection of **Claim 8** is incorporated; and Golden further discloses:

- parsing the markup language description and invoking each of the plurality of registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a*

init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked. ").

As per **Claim 10**, the rejection of **Claim 9** is incorporated; and Golden further discloses:

- generating output code using each of the plurality of invoked event handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output stream 15. "; Column 18: 12-13, "If the branching is not conditional, the two branches following the first engine work in parallel. ").*

As per **Claim 12**, the rejection of **Claim 1** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument. ")*; and

- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

As per **Claim 13**, the rejection of **Claim 12** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1: Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 14**, the rejection of **Claim 13** is incorporated; and Golden further discloses:

- creating a plurality of Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in a markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 16,** Golden discloses:

- a processor operable to execute computer program instructions *(see Column 7: 34-35, "... a computer system 10 with a processing unit and storage 11 for processing programs.")*;

- a memory operable to store computer program instructions executable by the processor *(see Column 7: 34-35, "... a computer system 10 with a processing unit and storage 11 for processing programs.")*; and

- computer program instructions stored in the memory and executable *(see Column 6: 25-28, "The disclosed embodiments of the computer program product comprise the disclosed program code which, for example, is stored on a computer-readable data carrier ...")* to perform the steps of:

- creating an event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.")*;

- registering the event handler *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.")*;

- parsing the markup language description and invoking the registered event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.")*; and

- generating output code using the invoked event handler *(see Column 14: 45-46, "... the taglet document is written to the output stream 15.")*.

However, Golden does not disclose:

- receiving an archive file to be deployed, wherein the archive file is an Enterprise Java Bean™ archive file;

- introspecting an input class included in the Enterprise Java Bean™ archive file to generate information relating to the input class; and

- generating a markup language description of the input class based on the generated information relating to the input class.

Sarkar discloses:

- receiving an archive file to be deployed, wherein the archive file is an Enterprise Java Bean™ archive file *(see Column 6: 50-55, "Then, at step 606, the Access Bean object AB701*

retrieves environment information about generic EJB 720, locates it, and creates an instance of the generic EJB 720 using standard EJB API calls."); and

- introspecting an input class included in the Enterprise Java Bean™ archive file to generate information relating to the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include receiving an archive file to be deployed, wherein the archive file is an Enterprise Java Bean™ archive file; and introspecting an input class included in the Enterprise Java Bean™ archive file to generate information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

Sijacic discloses:

- generating a markup language description of the input class based on the generated information relating to the input class *(see Paragraph [0119], "Once a Java class that implements the ISimpleWorkPerformer interface is created and compiled, an XML description file for the class is defined. The XML description file specifies the environment, input, and output*

parameters that the class uses. In addition, the XML file specifies some optional design parameters that may control the custom activity's appearance in the process builder 391.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sijacic into the teaching of Golden to include generating a markup language description of the input class based on the generated information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to describe a class of data objects using tags *(see Golden – Column 3: 40-48).*

As per **Claim 18**, the rejection of **Claim 16** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*; and

- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

As per **Claim 19**, the rejection of **Claim 18** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 20**, the rejection of **Claim 19** is incorporated; and Golden further discloses:

- creating a Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in a markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 27**, the rejection of **Claim 16** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain*

the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

As per **Claim 28**, the rejection of **Claim 27** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 29**, the rejection of **Claim 28** is incorporated; and Golden further discloses:

- creating a plurality of Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in a markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each*

tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").

As per **Claim 31**, Golden discloses:

- a computer readable medium *(see Column 6: 25-28, "... a computer-readable data carrier ...")*; and

- computer program instructions, recorded on the computer readable medium, executable by a processor, *(see Column 6: 25-28, "The disclosed embodiments of the computer program product comprise the disclosed program code which, for example, is stored on a computer-readable data carrier ...")* for performing the steps of:

- creating an event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.")*;

- registering the event handler *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.")*;

- parsing the markup language description and invoking the registered event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the*

taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked."); and

- generating output code using the invoked event handler *(see Column 14: 45-46, "... the taglet document is written to the output stream 15.").*

However, Golden does not disclose:

- receiving an archive file to be deployed, wherein the archive file is an Enterprise Java Bean™ archive file;

- introspecting an input class included in the Enterprise Java Bean™ archive file to generate information relating to the input class; and

- generating a markup language description of the input class based on the generated information relating to the input class.

Sarkar discloses:

- receiving an archive file to be deployed, wherein the archive file is an Enterprise Java Bean™ archive file *(see Column 6: 50-55, "Then, at step 606, the Access Bean object AB701 retrieves environment information about generic EJB 720, locates it, and creates an instance of the generic EJB 720 using standard EJB API calls.")*; and

- introspecting an input class included in the Enterprise Java Bean™ archive file to generate information relating to the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612,*

the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include receiving an archive file to be deployed, wherein the archive file is an Enterprise Java Bean™ archive file; and introspecting an input class included in the Enterprise Java Bean™ archive file to generate information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

Sijacic discloses:

- generating a markup language description of the input class based on the generated information relating to the input class *(see Paragraph [0119], "Once a Java class that implements the ISimpleWorkPerformer interface is created and compiled, an XML description file for the class is defined. The XML description file specifies the environment, input, and output parameters that the class uses. In addition, the XML file specifies some optional design parameters that may control the custom activity's appearance in the process builder 391.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sijacic into the teaching of Golden to include generating a markup language description of the input class based on the generated information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to describe a class of data objects using tags *(see Golden – Column 3: 40-48).*

As per **Claim 33**, the rejection of **Claim 31** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include

extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

As per **Claim 34**, the rejection of **Claim 33** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 35**, the rejection of **Claim 34** is incorporated; and Golden further discloses:

- creating a Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in a markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 37**, the rejection of **Claim 31** is incorporated; and Golden further discloses:

- creating a plurality of event handlers for a method node found in the markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML*

fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").

As per **Claim 38**, the rejection of **Claim 37** is incorporated; and Golden further discloses:

- registering each of the plurality of event handlers *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 39**, the rejection of **Claim 38** is incorporated; and Golden further discloses:

- parsing the markup language description and invoking each of the plurality of registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 40**, the rejection of **Claim 39** is incorporated; and Golden further discloses:

- generating output code using each of the plurality of invoked event handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 18:*

12-13, "If the branching is not conditional, the two branches following the first engine work in parallel.").

As per **Claim 42**, the rejection of **Claim 31** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar discloses:

- extracting information identifying methods included in the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

As per **Claim 43**, the rejection of **Claim 42** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 44**, the rejection of **Claim 43** is incorporated; and Golden further discloses:

- creating a plurality of Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in a markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 46**, the rejection of **Claim 5** is incorporated; and Golden further discloses:

- registering the created Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in a markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 6**, the rejection of **Claim 46** is incorporated; and Golden further discloses:

- parsing a markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the Simple Application Programming Interface for Extensible Markup Language event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 47**, the rejection of **Claim 20** is incorporated; and Golden further discloses:

- registering the created Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in a markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The*

application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").

As per **Claim 21**, the rejection of **Claim 47** is incorporated; and Golden further discloses:

- parsing a markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the Simple Application Programming Interface for Extensible Markup Language event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 22**, the rejection of **Claim 21** is incorporated; and Golden further discloses:

- creating a plurality of event handlers for a method node found in a markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 23**, the rejection of **Claim 22** is incorporated; and Golden further discloses:

- registering each of the plurality of event handlers *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 24**, the rejection of **Claim 23** is incorporated; and Golden further discloses:

- parsing a markup language description and invoking each of the plurality of registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 25**, the rejection of **Claim 24** is incorporated; and Golden further discloses:

- generating output code using each of the plurality of invoked event handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 18: 12-13, "If the branching is not conditional, the two branches following the first engine work in parallel.").*

As per **Claim 48**, the rejection of **Claim 35** is incorporated; and Golden further discloses:

- registering the created Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in a markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 36**, the rejection of **Claim 48** is incorporated; and Golden further discloses:

- parsing a markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the Simple Application Programming Interface for Extensible Markup Language event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 49**, the rejection of **Claim 44** is incorporated; and Golden further discloses:

- registering the plurality of created Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in a markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing*

process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").

As per **Claim 45**, the rejection of **Claim 49** is incorporated; and Golden further discloses:

- parsing a markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the plurality of Simple Application Programming Interface for Extensible Markup Language event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 50**, the rejection of **Claim 14** is incorporated; and Golden further discloses:

- registering the plurality of created Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in a markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 15**, the rejection of **Claim 50** is incorporated; and Golden further discloses:

- parsing a markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the plurality of Simple Application Programming Interface for Extensible Markup Language event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 51**, the rejection of **Claim 29** is incorporated; and Golden further discloses:

- registering the plurality of created Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in a markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 30**, the rejection of **Claim 51** is incorporated; and Golden further discloses:

- parsing a markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the plurality of Simple Application Programming Interface for Extensible Markup Language event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

Response to Arguments

19. Applicant's arguments with respect to Claims 1, 16, and 31 have been considered, but are moot in view of the new ground(s) of rejection.

In the remarks, Applicant argues that:

a) Sarkar discloses a method and system for running application code originally developed as simple Java Beans, in an Enterprise Java Bean (EJB) environment, without modifying the original application code. Thus, Sarkar discloses receiving a simple Java Beans file and introspecting the simple Java Beans. By contrast, claim 1, for example, requires receiving an archive file to be deployed, wherein the archive file is an Enterprise Java Bean archive file and introspecting an input class included in the Enterprise Java Bean archive file to generate information relating to the input class.

Examiner's response:

a) Sarkar discloses:

- receiving an archive file to be deployed, wherein the archive file is an Enterprise Java Bean™ archive file *(see Column 6: 50-55, "Then, at step 606, the Access Bean object AB701 retrieves environment information about generic EJB 720, locates it, and creates an instance of the generic EJB 720 using standard EJB API calls.")*; and

- introspecting an input class included in the Enterprise Java Bean™ archive file to generate information relating to the input class *(see Column 6: 66 and 67 to Column 7: 1-4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to retrieve the static variable of properties object C1 and obtain the name of the helper object it needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612, the generic EJB 720 uses Java reflection to get the main execution method of the helper object HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.")*.

Conclusion

20. The prior art made of record and not relied upon is considered pertinent to Applicant's disclosure.

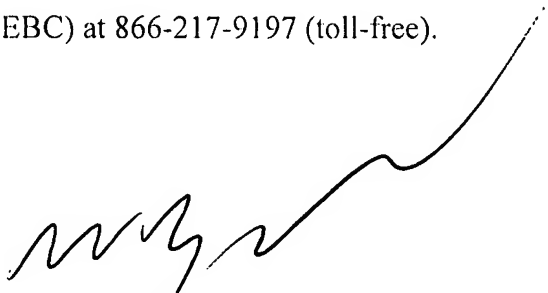
Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The

Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).



WEI ZHEN
SUPERVISORY PATENT EXAMINER

QC / QC
November 15, 2007